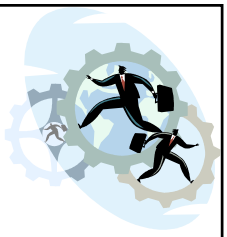


Model Driven Architecture – Best Practices

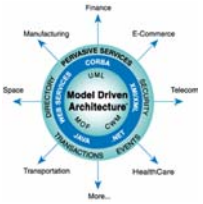
Am Beispiel der MDA-Suite XCoder

Dipl.-Inf. Constantin Szallies

LIANTIS IT-Consulting

Standardtechnologien der MDA

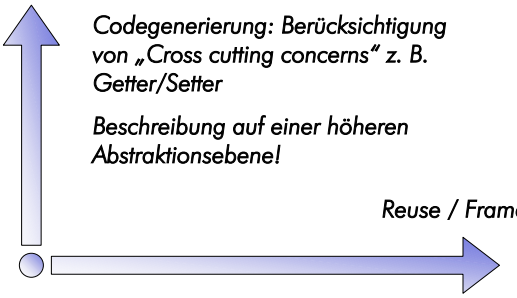
- Standardisiert von der Object Management Group (OMG)
- Unified Modeling Language (UML):
  - Standardisierte Modellierungssprache
- Meta Object Facility (MOF):
  - Wie werden Metamodelle beschrieben?
- XML Meta Data Interchange (XMI):
  - Austausch von Modellinformationen zwischen verschiedenen Werkzeugen
- Query / View / Transformation (QVT)
  - Metamodell-Transformationssprache



LIANTIS IT-Consulting

Warum MDA?

- Softwareentwicklung automatisieren!
- Zwei orthogonale Ansätze kombinieren:



Codegenerierung: Berücksichtigung von „Cross cutting concerns“ z. B. Getter/Setter

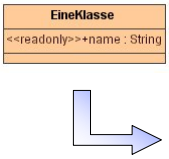
Beschreibung auf einer höheren Abstraktionsebene!

Reuse / Frameworks / Vererbung

LIANTIS IT-Consulting

Beispiel: Generierung von Gettern/Settern

- Nicht durch Vererbung automatisierbar!



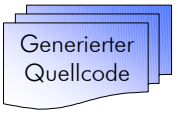

```
public class EineKlasse {
  private java.lang.String _name;
  private void setName(java.lang.String someObject) {
     _name = someObject;
  }
  public java.lang.String getName() {
     return _name;
  }
}
```


LIANTIS IT-Consulting

Softwareentwicklung mit der MDA

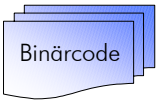
Modellierung

Generator



Individueller Code =
nicht automatisierte Bearbeitung

Compiler



MDA-Werkzeuge

- Zusammen mit UML-Werkzeug
  - ArcStyler (http://www.interactive-objects.com/)
  - iUML (http://www.kc.com/)
  - Ameos (http://www.aonix.com/)
- Add-on zum UML-Werkzeug
  - XCoder (http://sourceforge.net/projects/xcoder)
  - Open ArchitectureWare (http://sourceforge.net/projects/architecturware)

Die XCoder-Suite

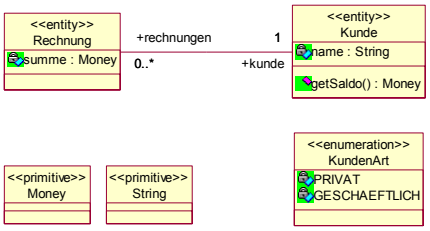
- Wiederverwendbare Komponenten und anpassbare Standard-Generatoren
- Open Source: http://sourceforge.net/projects/xcoder
- In Projekten von Liantis verwendet und weiterentwickelt, wenn Standard-Lösungen ungeeignet
- Das komplette Framework ist vollständig modelliert und generiert sich selbst: MDA²
- Leichtgewichtig, schnell und einfach zu erweitern

LIANTIS IT-Consulting

XCoder-Architektur (hier: für C++)

LIANTIS IT-Consulting

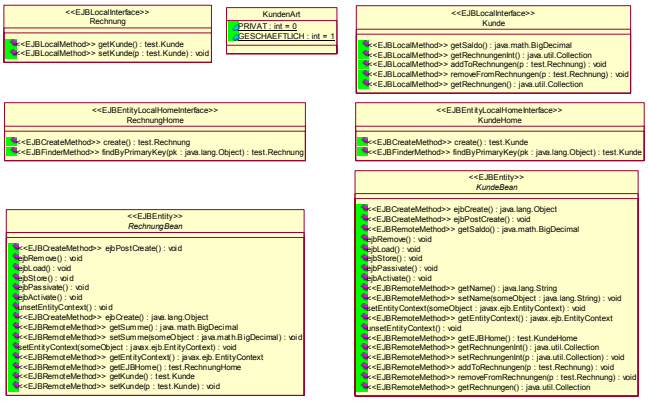
Platform Independent Model (PIM)



LIANTIS IT-Consulting

Platform Specific Model (PSM)

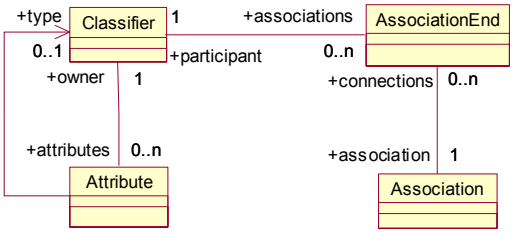
- Durch Transformation des PIM erzeugt



LIANTIS IT-Consulting

UML-Metamodell

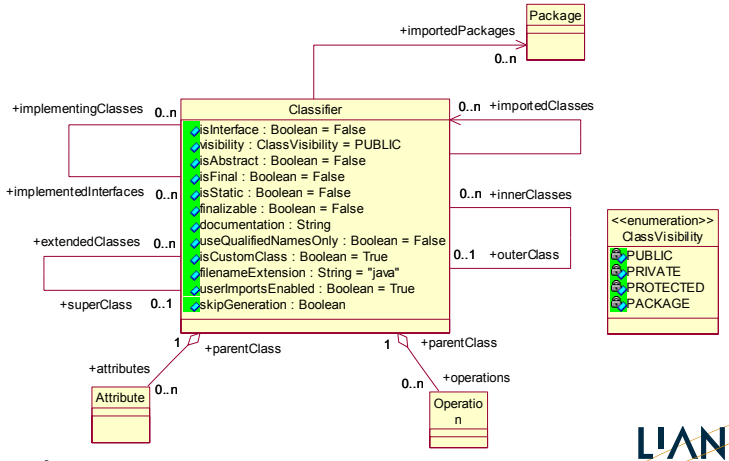
- Eine Instanz des Metamodells ist das im UML-Werkzeug erstellte Modell
- Einlesen des Modells im XMI-Format



LIANTIS IT-Consulting

Java-Metamodell

- Metamodell der Programmiersprache Java



LIANTIS IT-Consulting

Best Practice:
Konsequentes Forward Engineering

- Modellierung
- Generierung
- Ergänzen des individuellen Codes

Modell und Code jederzeit synchron!

Roundtrip Engineering bedeutet: Modell und Code äquivalent

Ermöglicht die Erstellung eines Modells auf höherer Abstraktionsebene!

LIANTIS IT-Consulting

Best Practice:
Generierter Code gehört dem Generator

- Wartung des Generators statt Wartung des generierten Codes
- Generierter Code darf nur vom Generator geändert werden
- Relative Wartungskosten ≅ LOC / Funktionalität
- Mit Generierung:
  - (Individuelle LOC + Generator LOC) / Funktionalität
- Gegenbeispiel: Code-“Wizard“

LIANTIS IT-Consulting

Best Practice: Nicht um jeden Preis alles generieren

- Anfangen, wo Kosten klein und Nutzen groß
  - Statische Diagramme (z.B. Klassendiagramm)
- Dynamische Diagramme sind aufwendiger
  - Zustandsübergangs-Diagramme
  - Aktivitäts-Diagramme
- Nutzlos für die Generierung
  - Use-Case-Diagramme
  - Sequenz-Diagramme
- Mischen von generiertem und individuellem Code durch Einfügepunkte
- Vollständige Generierung möglich: Executable UML mit der Object Constraint Language (OCL)

LIANTIS IT-Consulting

Beispiel: Einfügepunkte

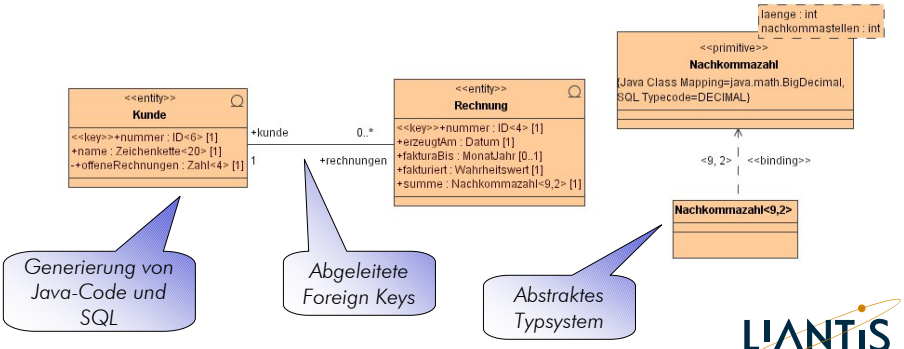
- Geschützte Bereiche im generierten Code bleiben bei wiederholter Generierung erhalten

```
// Der Umsatz eines Kunden berechnet sich aus der Summe der
// Rechnungssummen aller seiner Rechnungen.
Base::Money KundeImpl::umsatzBerechnen() const throw (Base::TechnicalException)
{
    // @BEGINPROTECT 39BDE46C005D
    RechnungSet::const_iterator iter;
    RechnungSet rechnungen = this->getRechnungen();
    Base::Money summe = 0.0;
    for ( iter = rechnungen.begin(); iter != rechnungen.end(); ++iter )
    {
       summe += (*iter)->getRechnungssumme();
    }
    return summe;
    // @ENDPROTECT
}
```


LIANTIS IT-Consulting

Best Practice:
Modell auf hoher Abstraktionsebene (PIM)

- Design-Entscheidungen automatisieren
- Mehrere zusammengehörige Artefakte generieren
- Unabhängiger von technischen Änderungen



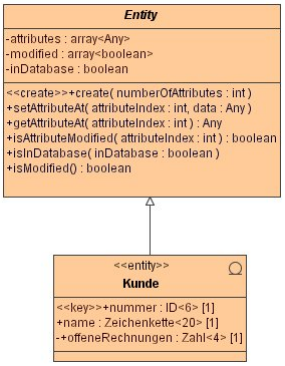
LIANTIS IT-Consulting

Best Practice: Generierung zusammen mit Architektur optimieren

- Anwendungsentwickler stellt Anforderungen an die „Usability“ der Architektur
  - Einfaches Bugfixing
  - Einfach zu verwenden
  - Einfache Wartbarkeit
- Eine generierungsfähige Architektur braucht dies nicht!
- Beispiel: Leichtgewichtige Persistenzschicht
  - Nur 6 Klassen
  - Für manuelle Programmierung nicht praktikabel

LIANTIS IT-Consulting

Beispiel: Leichtgewichtige Persistenzschicht

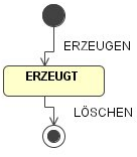



```
public final class Kunde extends architecture.persistency.Entity
{

  public Kunde() {
    super(3);
  }

  public java.lang.Integer getNummer()
  {
    return (java.lang.Integer)this.getAttributeAt(0);
  }

  public void setNummer(java.lang.Integer someObject)
  {
    this.setAttributeAt(0,someObject);
  }
  ...
}
```


LIANTIS IT-Consulting

Best Practice: Generierung von Metamodell-Informationen

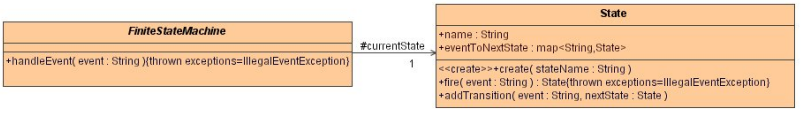
- Trick, um Generierungsverfahren zu vereinfachen
  - Keine Generierung von Businesslogik-Quellcode
  - Sondern Generierung von Metainformationen für eine Softwarearchitektur
- Beispiel: Generierung für Zustandsautomaten



LIANTIS IT-Consulting

Beispiel: Generierung von Zustandsautomaten

- Architektur:



- Generierter Code:

```
public final class MyStateMachine extends FiniteStateMachine
{
  public MyStateMachine() {
    State state_START = new State("START");
    State state_ERZEUGT = new State("ERZEUGT");
    State state_GELÖSCHT = new State("GELÖSCHT");

    state_START.addTransition("ERZEUGEN",state_ERZEUGT);
    state_ERZEUGT.addTransition("LÖSCHEN",state_GELÖSCHT);

    this.setCurrentState(state_START);
  }
}
```

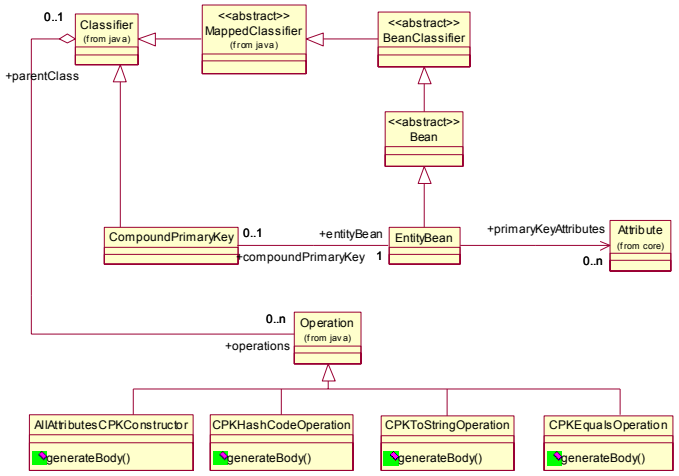
Zustände erzeugen

Übergänge erzeugen

Best Practice: Architektur-Metamodellierung durch PSM-Erweiterung

- Für jede Softwarearchitektur kann ein Architektur-Metamodell erstellt werden
- Architektur-Metamodell ist Erweiterung des Metamodells der verwendeten Programmiersprache
- Architektur-Metamodell ist das PSM des Generators
- Beispiel: EJB Beans als Erweiterung des Java-Metamodells

Beispiel: EJB-Metamodell als Erweiterung des Java-Metamodells



LIANTIS IT-Consulting

Best Practice: Codegenerierung mit Templates

- Templates enthalten eins zu eins den zu generierenden Quellcode (Zielsprache)
- Einbettung von Befehlen der Templatesprache
  - Verzweigungen
  - Schleifen
  - Zugriff auf Metamodell-Informationen
- Beispiel: XCoder-Template zur Generierung der ‚toString'-Methode einer EJB-Primärschlüsselklasse

LIANTIS IT-Consulting

Beispiel: XCoder Template

```
public void generateBody(com.liantis.io.Printer printer)
 {
   // @BEGINPROTECT _3C8E2EF60260
   java.util.Vector keys = ((CompoundPrimaryKey)this.getParentClass()).
                           getEntityBean().getPrimaryKeyAttributes();

   Attribute firstAttribute = (Attribute)keys.elementAt(0);

   [
   return "(" + [[firstAttribute.getName()]] +
   ]
   for(int i=1,j=keys.size();i<j;i++)
   {
     Attribute anAttribute = (Attribute)keys.elementAt(i);
     [
       ", " + [[anAttribute.getName()]] +
     ]
   }
   [
     ")";
   ]
   // @ENDPROTECT
 }
```

Umwandlung mittels Präprozessor

Problembereich: Arbeiten in großen Teams

- Ohne MDA: Datei-basiertes Arbeiten
  - Sperren von Dateien
  - Automatisches Zusammenführung von gleichzeitigen Änderungen
  - Trennung zwischen Schnittstelle und Implementierung
- Mit MDA: Modell-basiertes Arbeiten
  - XMI: Keine Modularisierung
  - Toolspezifische Lösungen
    - Rose CAT-Dateien
    - MagicDraw Teamwork-Server
  - Komponenten-Konzept auf Modell-Ebene

MDA-Projekt 1

- Abrechnungssystem für Telekommunikationsunternehmen, Energieversorger und Betreiber von Kundenbindungssystemen
- Gemeinsam verwendete Basis für alle Branchen
- UML-Werkzeug: Rational Rose
- Produktiv seit 1998
- Bestehende bewährte C++-Architektur
  - Plattformunabhängig (Unix / Windows)
  - Über viele Jahre entwickelt und optimiert
  - hochperformantes Persistenzframework
  - Massendatenverarbeitung

LIANTIS IT-Consulting

MDA-Projekt 1: Lessons learned

- Geringer Aufwand bei der MDA-Einführung ist möglich (20 PT)
- Erreichen des ROI schon im ersten Projekt nach wenigen Wochen
- Auch kleine und mittelgroße Projekte können MDA sinnvoll einsetzen
- Existierende Umgebungen und laufende Projekte behindern nicht die erfolgreiche Einführung

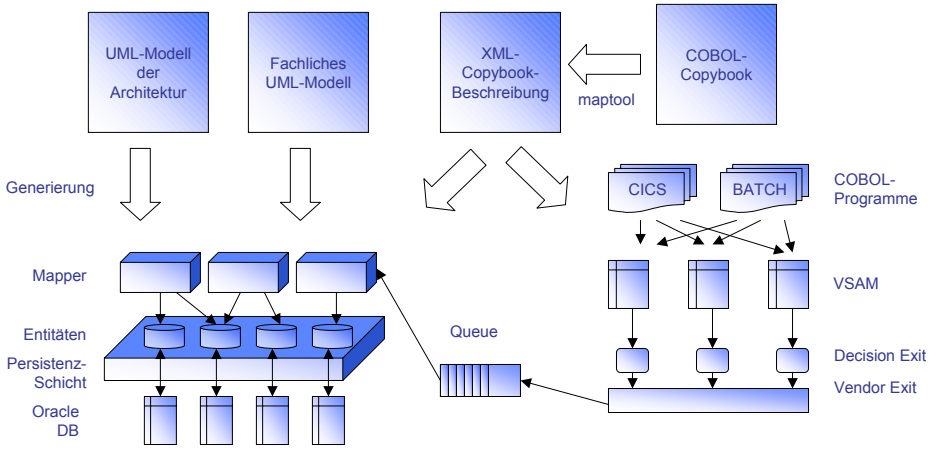
LIANTIS IT-Consulting

MDA-Projekt 2

- Spiegelung von Daten eines HOST-Systems in einer relationalen Datenbank
- Generierung von Java aus UML
- Generierung von Java und Assembler aus einer XML-Beschreibung
- Anpassung des XCoder/J + Coaching: 15 MT
- UML-Werkzeug: MagicDraw

LIANTIS IT-Consulting

MDA-Projekt 2: Architektur



LIANTIS IT-Consulting

Unsere Kompetenz im Bereich MDA

- 1997
- 1998 - 1999
- 1999
- 1999 - 2001
- 2001 - ...
- 2002 - 2003
- 2004

....weiter

1. Einsatz v. MDA-Standardwerkzeugen
2. Entwicklung v. MDA-Individuallösungen

- Modellgetriebene Entwicklung seit 1997
  - COBOL/Java: Versicherungsbranche
  - COBOL: Finanzdienstleistungsbranche
  - Java/EDIFACT: Standardisierungsorg.
  - C++: Immobilienwirtschaft
  - Java/C++/C#/J2EE: SE-Werkzeuge
  - C++: Telekommunikation
  - J2EE: Energiewirtschaft
  - J2EE: Bank

LIANTIS IT-Consulting

Free your work.

Durch:

Constantin Szallies
constantin.szallies@liantis.com

Liantis GmbH & Co. KG
St.-Anton-Straße 69 - 71
47798 Krefeld
Fon: 0 21 51 / 931 86-60
Fax: 0 21 51 / 931 86-61
info@liantis.com
www.liantis.com

LIANTIS IT-Consulting